摘要

Feature engineering has been the key to the success of many prediction models. However, the process is nontrivial and often requires manual feature engineering or exhaustive searching. DNNs are able to automatically learn feature interactions; however, they generate all the interactions implicitly, and are not necessarily efficient in learning all types of cross features. In this paper, we propose the Deep & Cross Network (DCN) which keeps the benefits of a DNN model, and beyond that, it introduces a novel cross network that is more efficient in learning certain bounded-degree feature interactions. In particular, DCN explicitly applies feature crossing at each layer, requires no manual feature engineering, and adds negligible extra complexity to the DNN model. Our experimental results have demonstrated its superiority over the state-of-art algorithms on the CTR prediction dataset and dense classification dataset, in terms of both model accuracy and memory usage.

1. 介绍

> DCN全称Deep & Cross Network。

CTR预估全称是*Click Through Rate*，就是展示给用户的广告或者商品，估计用户点击的概率。公司规模较大的时候，CTR直接影响的价值在数十亿美元的级别。广告支付一个非常流行的模型就是CPC(cost-per-click)，就是按照用户的点击来付钱。那么准确的进行CTR预估，展现给用户他们最可能点击的广告就非常重要了。

传统的CTR预估模型需要大量的特征工程，耗时耗力；引入DNN之后，依靠神经网络强大的学习能力，可以一定程度上实现自动学习特征组合。但是DNN的缺点在于隐式的学习特征组合带来的不可解释性，以及低效率的学习(并不是所有的特征组合都是有用的)。

DCN全称*Deep & Cross Network*，是谷歌和斯坦福大学在2017年提出的用于Ad Click Prediction的模型。DCN(Deep Cross Network)在学习特定阶数组合特征的时候效率非常高，而且同样不需要特征工程，引入的额外的复杂度也是微乎其微的。

2. 相关工作

最开始FM使用隐向量的内积来建模组合特征；FFM在此基础上引入field的概念，针对不同的field上使用不同隐向量。但是，这两者都是针对低阶的特征组合进行建模的。 随着DNN在计算机视觉、自然语言处理、语音识别等领域取得重要进展，DNN几乎无限的表达能力被广泛的研究。同样也尝试被用来解决web产品中输入高维高稀疏的问题。DNN可以对高维组合特征进行建模，但是DNN是否就是针对此类问题最高效的建模方式那？直到现在，业界也没有一个准确的答案。 在Kaggle上的很多比赛中，大部分的获胜方案都是使用的人工特征工程，构造低阶的组合特征，这些特征意义明确且高效。而DNN学习到的特征都是高度非线性的高阶组合特征，含义非常难以解释。那么是否能设计一种DNN的特定网络结构来改善DNN，使得其学习起来更加高效那？

业内进行了很多探索，DCN就是其中一个。

3. DCN特点

DCN特点如下：

1. 使用cross network，在每一层都应用feature crossing。高效的学习了*bounded degree*组合特征。不需要人工特征工程。
2. 网络结构简单且高效。多项式复杂度由*layer depth*决定。
3. 相比于DNN，DCN的logloss更低，而且参数的数量将近少了一个数量级。

4. DCN

还记得DCN的全称是什么吗？*Deep & Cross Network*，聪明的你一定答对了吧！下面就跟着小编一起就进入到DCN里面一探究竟吧。

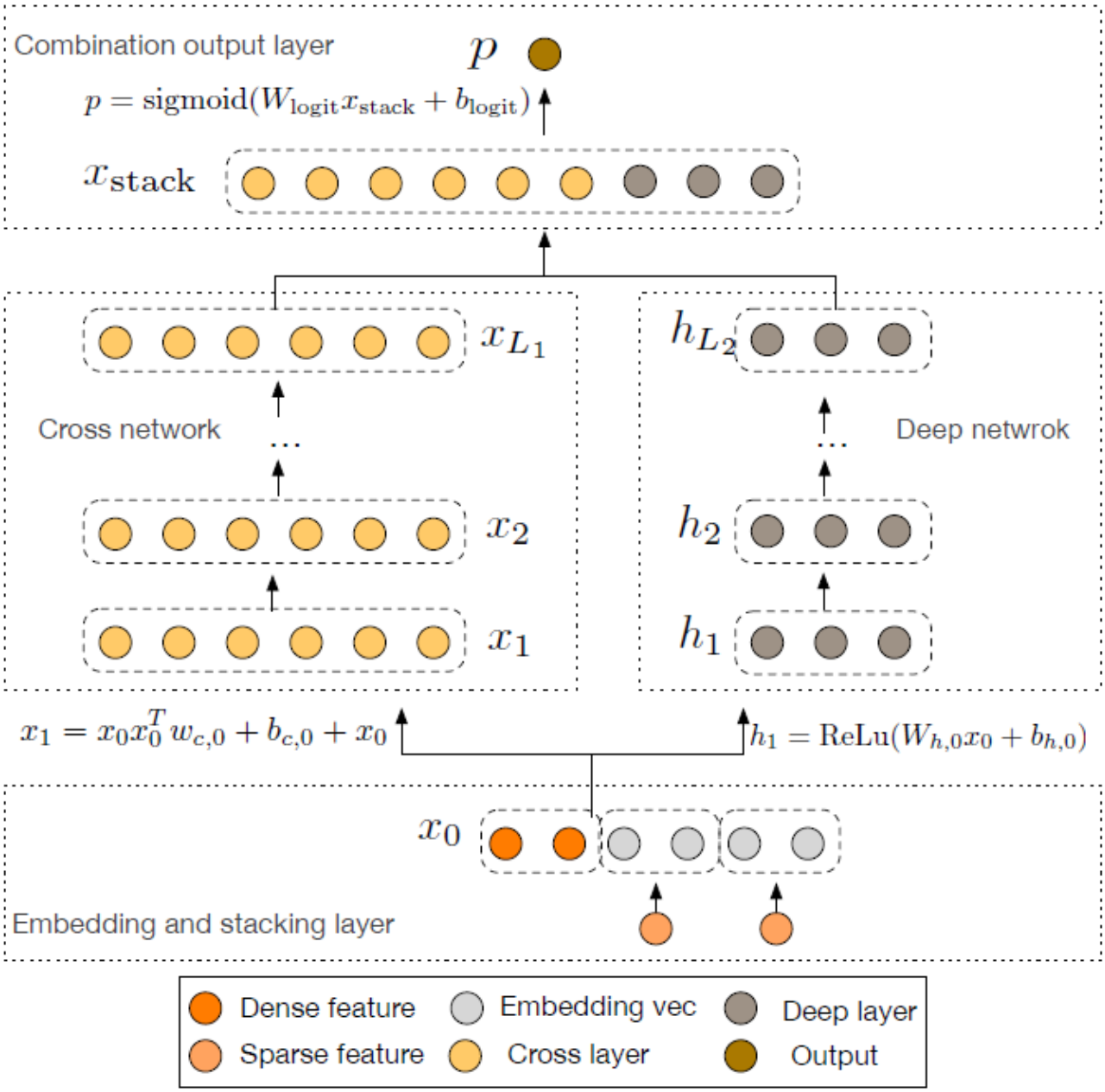

Figure 1: The Deep & Cross Network

DCN架构图如上图所示：最开始是 **Embedding and stacking layer**，然后是并行的 **Cross Network** 和 **Deep Network**，最后是 **Combination Layer** 把 **Cross Network** 和 **Deep Network** 的结果组合得到 **Output**。

4.1 Embedding and Stacking Layer

这一层说起来其实非常的简单，就两个功能*Embed*和*Stack*。

Embedding：

在网络规模推荐系统的CTR预测任务中，输入主要是分类特征，通常的处理办法就是*one-hot*，但是one-hot之后输入特征维度非常高非常系数，如“country=usa”。这些特征通常是编码为独热向量如`[0,1,0]`；然而，这通常会产生超高维度的特征空间。 所以有了**Embedding** 来大大的降低输入的维度，**就是把这些 binary features 转换成 dense vectors with real values**（通常称为嵌入向量）：

$$x_{embed,i} = W_{embed,i} x_i$$

其中 $x_{embed,i}$ 是embedding vector，x_i 是第 i 个 category 的二元输入，$W_{embed,i} \in R^{n_e \times n_v}$ 是对应的 embedding matrix，会与网络中的其它参数一起进行优化，n_e, n_v 分别是embedding size和vocabulary size。

Embedding 操作其实就是用一个矩阵和 one-hot 之后的输入相乘，也可以看成是一次查询（lookup）。这个Embedding矩阵跟网络中的其他参数是一样的，是需要随着网络一起学习的。

Stacking 处理完了类别型特征，还有连续型特征没有处理那。所以我们把连续型特征规范化之后，和嵌入向量 **stacking** 到一起，就得到了原始的输入：

$$x_0 = [x_{embed,1}^T, \dots, X_{embed,k}^T, X_{dense}^T]。$$

这一部分在tensorflow中，使用`tf.feature_column`API可以很容易实现，大致代码结构如下：

```
embed0 = tf.feature_column.embedding_column(...)
...
dense0 = tf.feature_column.indicator_column(...)
dense1 = tf.feature_column.numeric_column(...)
...
columns = [embed0, ..., dense0, dense1, ...]
x0 = tf.feature_column.input_layer(features, feature_columns)
```

4.2 Cross Network

交叉网络的核心思想是以有效的方式应用显式特征交叉。交叉网络由交叉层组成，每个层具有以下公式：

$$x_{l+1} = x_0 x_l^T w_l + b_l + x_l = f(x_l, w_l, b_l) + x_l$$

其中:

- x_l, x_{l+1} 是列向量（column vectors），分别表示来自第 l 层和第 $(l+1)$ 层cross layers的输出；
- $w_l, b_l \in R^d$ 是第 l 层 layer 的 weight 和 bias 参数。

在完成一个特征交叉 f 后，每个 cross layer 会将它的输入加回去，对应的mapping function $f: R^d \to R^d$，刚好等于残差 $x_{l+1} - x_l$，这里借鉴了残差网络的思想。

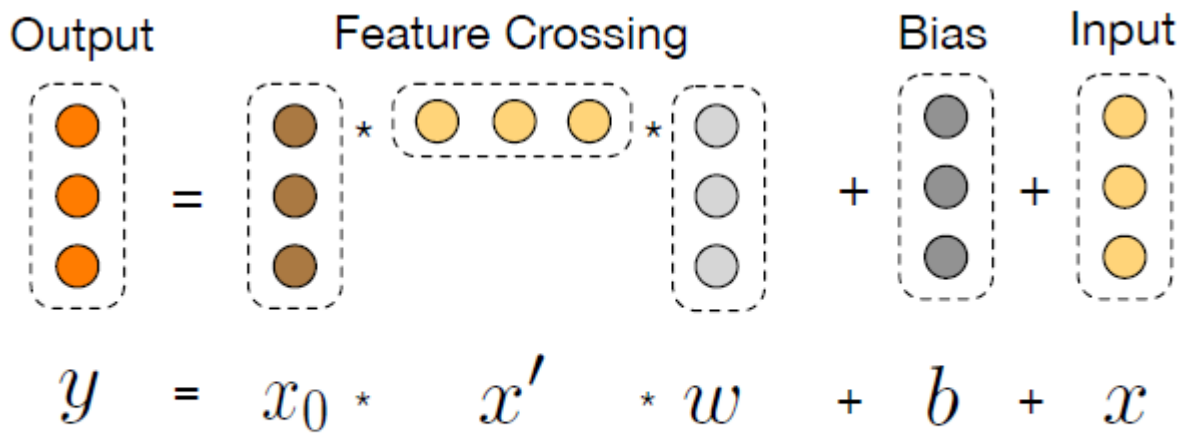

Figure 2: Visualization of a cross layer.

特征的高阶交叉（**high-degree interaction**）：

cross network的独特结构使得交叉特征的阶（the degress of cross features）随着layer的深度而增长。对于第 l 层 layer，它的最高多项式阶（在输入x_0 上）是 $l+1$。 实际上，cross network由这些交叉项 $x_1^{\alpha_1} x_2^{\alpha_2} \ldots x_d^{\alpha_d}$ 组成，对应的阶从 1 到 $l+1$。

复杂度分析：

假设 L_c 表示 cross layers 的数目，d 表示输入 x_0 的维度。那么，在该cross network中涉及的参数数目为：

$$d \times L_c \times 2$$

因为每一层的W和b都是d维度的。从上式可以发现，复杂度是输入维度d的线性函数。所以相比于deep network，cross network引入的复杂度微不足道。这样就保证了DCN的复杂度和DNN是一个级别的。论文中表示，**Cross Network之所以能够高效的学习组合特征，就是因为x0 * xT的秩为1(rank-one 特性(两个向量的叉积))，使得我们不用计算并存储整个的矩阵就可以得到所有的cross terms**。

主流的实现cross layer的方法，代码如下:

```
def cross_layer(x0, x, name):
  with tf.variable_scope(name):
    input_dim = x0.get_shape().as_list()[1]
    w = tf.get_variable("weight", [input_dim],
initializer=tf.truncated_normal_initializer(stddev=0.01))
    b = tf.get_variable("bias", [input_dim],
initializer=tf.truncated_normal_initializer(stddev=0.01))
    xx0 = tf.expand_dims(x0, -1)  # shape <?, d, 1>
    xx = tf.expand_dims(x, -1)  # shape <?, d, 1>
    mat = tf.matmul(xx0, xx, transpose_b=True)  # shape <?, d, d>
    return tf.tensordot(mat, w, 1) + b + x  # shape <?, d>
```

这种方法在逻辑上没有什么问题，但实际上却是**非常消耗计算和存储资源**的，原因在于显式地计算 $x_0 x_l^T$ 需要非常大的内存空间来存储临时计算结果。我们来计算一下，一个 cross layer 仅仅是计算 $x_0 x_l^T$ 这一个操作就需要消耗 $batch\_size \times d \times d \times 4$ 字节的内存（一个浮点数占4个字节）。在企业级的模型中，d 通常是几千甚至几万的量级，假设 $d = 1k$，则需要 $batch\_size \times 4M$ 的存储空间，这通常情况下已经是 G 级别的大小了，何况我们仅仅计算了一个 Layer，别忘了我们总共有 L_c 个 cross layer。另外，该操作的结果（一个矩阵）再和 w 向量相乘时也是非常消耗计算资源的。即使你在离线训练时通过减少 cross layer 的个数，减小 batch_size 等手段完成了模型的训练，在模型部署中线上之后，线性的打分系统依然要面临 Out of Memory 的风险，因为线上预测我们总是希望一次请求尽可能返回多条记录的预测分数，否则要么是影响全局的效果，要么是需要更多的请求次数，从而面临巨大的性能压力。

正确的实现方式不是先计算 $x_0 x_l^T$，而是先计算 $x_l^T w$，因为 $x_l^T w$ 的计算结果是一个标量，几乎不占用存储空间。这两种方法的计算结果是一致的，因为矩阵乘法是满足结合律的：`(AB)C=A(BC)`。高效的实现代码如下：

```
def cross_layer2(x0, x, name):
  with tf.variable_scope(name):
    input_dim = x0.get_shape().as_list()[1]
    w = tf.get_variable("weight", [input_dim],
initializer=tf.truncated_normal_initializer(stddev=0.01))
    b = tf.get_variable("bias", [input_dim],
initializer=tf.truncated_normal_initializer(stddev=0.01))
    xb = tf.tensordot(tf.reshape(x, [-1, 1, input_dim]), w, 1)
    return x0 * xb + b + x
```

在上面的实现中，我们使用了 `tf.reshape` 操作实现了 x_l 的转置，因为 x_l 实际上是一个向量，并不是一个矩阵，因此这种方法是可行的。下面给出构建整个交叉网络的tensorflow代码：

```
def build_cross_layers(x0, params):
  num_layers = params['num_cross_layers']
  x = x0
  for i in range(num_layers):
    x = cross_layer2(x0, x, 'cross_{}'.format(i))
  return x
```

泛化**FM**

跟FM一样，DCN同样也是基于参数共享机制的，参数共享不仅仅使得模型更加高效而且使得模型可以泛化到之前没有出现过的特征组合，并且对噪声的抵抗性更加强。 在FM模型中，特征 x_i 和权重向量 v_i 相关联，交叉项 $x_i x_j$ 的权重由 $< v_i, v_j >$ 在DCN模型中，x_i 和标量 $\{w_k^{(i)}\}_{k=1}^l$ 相关联，并且 $w_i x_j$ 的权重从集合 $\{w_k^{(i)}\}_{k=1}^l$ 和 $\{w_k^{(j)}\}_{k=1}^l$ 中计算而来。两种模型的每个特征学习独立于其他特征的参数，交叉项的权重是对应参数的特定组合。 参数共享不仅使模型更有效,但也使模型可以泛化出不可见的特征组合，使得模型更健壮。例如,以具有稀疏特征的数据集，如果两个二进制特征 x_i 和 x_j 在训练数据集中很少或从不共现，如 $x_i \neq 0 \wedge x_j \neq 0$，然后 $x_i x_j$ 学到的权重对于预测也没什么意义。

FM是一个非常浅的结构，并且限制在表达二阶组合特征上，DeepCrossNetwork(DCN)把这种参数共享的思想从一层扩展到多层，并且可以学习高阶的特征组合 $x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_d^{\alpha_d}$。但是和FM的高阶版本的变体不同，DCN的参数随着输入维度的增长是线性增长的。

有效映射

对于cross layer可以换一种理解方式。假设 $\tilde{x} \in R^d$ 是一个cross layer的输入，cross layer首先构建 d^2 个关于 $x_i\tilde{x}_j$ 的 pairwise 交叉，接着以一种内存高效的方式将它们投影到维度 d 上。如果采用全连接 Layer 那样直接投影的方式会带来3次方的开销。Cross layer提供了一种有效的解决方式，将开销减小到维度 d 的量级上：考虑到 $x_p = x_0\tilde{x}^T w$ 等价于：

$$x_p^T = [x_1\tilde{x}_1 \dots x_1\tilde{x}_d \dots x_d\tilde{x}_1 \dots x_d\tilde{x}_d] \begin{bmatrix} w & 0 & \dots & 0 \\ 0 & w & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & w \end{bmatrix}$$

其中，行向量包含了所有 d^2 个关于 $x_i\tilde{x}_j$ 的 pairwise 交叉，投影矩阵具有一个块对角化结构，其中 $w \in R^d$ 是一个列向量。

值得注意的是，正是因为cross network的参数比较少导致它的表达能力受限，为了能够学习高度非线性的组合特征，DCN并行的引入了Deep Network。

4.3 Deep Network

交叉网络的参数数目少，从而限制了模型的能力（capacity）。为了捕获高阶非线性交叉，我们平行引入了一个深度网络。

深度网络就是一个全连接的前馈神经网络，每个深度层具有如下公式：

$$h_{l+1} = f(W_l h_l + b_l)$$

其中：

- $h_l \in R^{n_l}, h_{l+1} \in R^{n_{l+1}}$ 分别是第 l 层和第 (l+1) 层hidden layer；
- $W_l \in R^{n_{l+1} \times n_l}, b_l \in R^{n_{l+1}}$ 第 l 层 deep layer 的参数；
- $f(\cdot)$ 是ReLU function。

复杂度分析：出于简洁性，我们假设所有的deep layers具有相同的size。假设LdLd表示deep layers的数目，m表示deep layer的size。那么，在该deep network中的参数的数目为：

$$d \times m + m + (m^2 + m) \times (L_d - 1)$$

其中 $d \times m + m$ 是第一层参数，而第二层至最后一层参数为：$(m \times m + m) \times (L_d - 1)$，因为到了第二层，输入已经转变成了 m 维。

```
def build_deep_layers(x0, params):
  # Build the hidden layers, sized according to the 'hidden_units' param.
  net = x0
  for units in params['hidden_units']:
    net = tf.layers.dense(net, units=units, activation=tf.nn.relu)
  return net
```

4.4 Combination Layer

Combination Layer 把 Cross Network 和 Deep Network 的输出拼接起来，然后经过一个加权求和后得到 logits，然后经过 sigmoid 函数得到最终的预测概率。形式化如下：

$$p = \sigma([x_{L_1}^T, h_{L_2}^T]w_{logits})$$

p 是最终的预测概率；X_{L1} 是 d 维的，表示 Cross Network 的最终输出；h_{L2} 是 m 维的，表示 Deep Network 的最终输出；W_{logits} 是 Combination Layer 的权重；最后经过 sigmoid 函数，得到最终预测概率。

损失函数使用**带正则项的 log loss**，形式化如下：

$$loss = -\frac{1}{N}\sum_{i=1}^{N} y_i log(p_i) + (1-y_i)log(1-p_i) + \lambda \sum_{l} ||w||^2$$

另外，针对Cross Network和Deep Network，DCN是**一起训练的**，这样网络可以知道另外一个网络的存在。

类似于WDL模型，我们对两个network进行jointly train，在训练期间，每个独立的network会察觉到另一个。下面给出整个模型的实现代码：

```
def dcn_model_fn(features, labels, mode, params):
  x0 = tf.feature_column.input_layer(features, params['feature_columns'])
  last_deep_layer = build_deep_layers(x0, params)
  last_cross_layer = build_cross_layers(x0, params)
  last_layer = tf.concat([last_cross_layer, last_deep_layer], 1)
  my_head = tf.contrib.estimator.binary_classification_head(thresholds=[0.5])
  logits = tf.layers.dense(last_layer, units=my_head.logits_dimension)
  optimizer = tf.train.AdagradOptimizer(learning_rate=params['learning_rate'])
  return my_head.create_estimator_spec(
    features=features,
    mode=mode,
    labels=labels,
    logits=logits,
    train_op_fn=lambda loss: optimizer.minimize(loss,
global_step=tf.train.get_global_step())
  )
```

总结

DCN主要有以下几点贡献：

- 提出一种新型的交叉网络结构，可以用来提取交叉组合特征，并不需要人为设计的特征工程；
- 这种网络结构足够简单同时也很有效，可以获得随网络层数增加而增加的多项式阶（polynomial degree）交叉特征；
- 十分节约内存（依赖于正确地实现），并且易于使用；
- 实验结果表明，DCN相比于其他模型有更出色的效果，与DNN模型相比，较少的参数却取得了较好的效果。

参考文献

[1] Deep & Cross Network for Ad Click Prediction

[2] Deep&Cross Network模型理论和实践

[3] 谷歌DCN模型理论与实践

[4] 距离玩转企业级DCN(Deep & Cross Network)模型，你只差一步